Proceedings of 10th IOE Graduate Conference
Peer Reviewed
ISSN: 2350-8914 (Online), 2350-8906 (Print)
Year: 2021 Month: October Volume: 10

Performance Analysis of Shard Selection Techniques on Elasticsearch

Yashasvi Raj Pant [a], Basanta Joshi [b], Nanda Bikram Adhikari [c]

[a, b, c] *Department of Electronics and Computer Engineering, Pulchowk Campus, IOE, Tribhuvan University, Nepal*
Corresponding Email: [a] 075mscsk020.yashasvi@pcampus.edu.np , [c] adhikari@ioe.edu.np

Abstract
Distributed systems typically consist of several nodes connected together for handling search operations. Data is divided into those nodes for the purpose of parallel processing and replications. Elasticsearch is the popular distributed search engine where data is organized into indices. Each index of Elasticsearch consists of one or more shards and those shards can be distributed over different nodes. When a search operation is performed on a particular index, sending the search requests to all the related shards distributed over different nodes might result in high latency especially when the size of the cluster is large and nodes are far apart. Shard Selection is the technique that attempts to forward the query to the highly relevant shards discarding other non-relevant shards and thus decreasing the latency. Shard selection comes with the cost of relevance, it's obvious that the application of the shard selection algorithm might decrease the query relevance. There are several shard selection algorithms developed time and again. Among them, ReDDe, Sushi, and Rank-S are very popular. In this paper, implementation of those three algorithms, performance analysis along with the optimization of shard-related parameters are done.
The experimentation is performed using Insider Threat Test Dataset(CERT V6.2) collected from Carnegie Mellon University site. In terms of average latency, Rank-S is performing 14.92% and 9.83% better than SUSHI and ReDDe respectively. Similarly, in terms of Average Document Score, Rank-S is performing 21.68% and 5.488% better than SUSHI and ReDDe respectively.

Keywords
Nodes, Elasticsearch, Index, Shards, ReDDe, Sushi, Rank-S

1. Introduction

Parallel processing is the core of any distributed system. To perform parallel processings, a distributed system consists of several nodes linked across a network. To increase the performance of the distributed system, data is divided into chunks across those nodes which are called shards. Hence, sharding is the technique to divide the data into chunks creating partitions of the data and distributing it across the nodes for the purpose of parallelism and replications [1].

Elasticsearch is one of the distributed, Lucene-based, scalable, open-source search engines where sharding is used. It is capable of performing RESTful CRUD operations in a distributed manner for a huge amount of data.

Shard is the core of the distributed system of Elasticsearch. There are two types of shards in Elasticsearch: primary shard(commonly referred to as Shard) and replica shard(commonly referred to as Replica). Each shard is a single Lucene Index of Elasticsearch.

Each document of the Elasticsearch belongs to a primary shard whereas a replica shard is the copy of the primary shard. Elasticsearch is actually a grouping of one or more shards and the shard is actually a self-contained index.

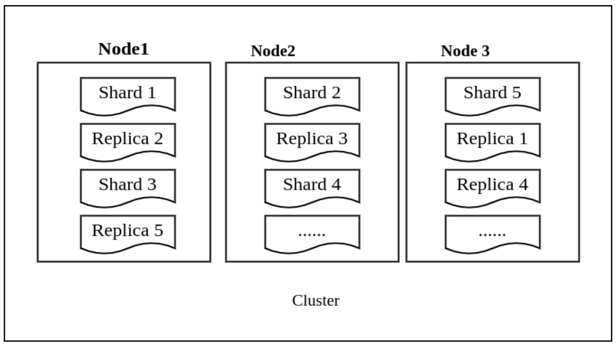

Figure 1: Figure showing the distribution of primary and replica shards of an index over different nodes

Shard selection is one of the optimization techniques for distributed search engines like Elasticsearch. The key concept of the shard selection technique is to process the user query only to the shard that contains relevant user documents while ignoring others. Shard selection typically consists of the concept of broker nodes and data nodes. A data node stores data, process search queries and return the relevant document. Each data node contains one or more shards. A broker node is responsible for receiving the search query from a user, re-routing the query to the appropriate data nodes, receive the result from those nodes, combine and return it to the user. The concept of Shard Selection is shown in the figure 2.

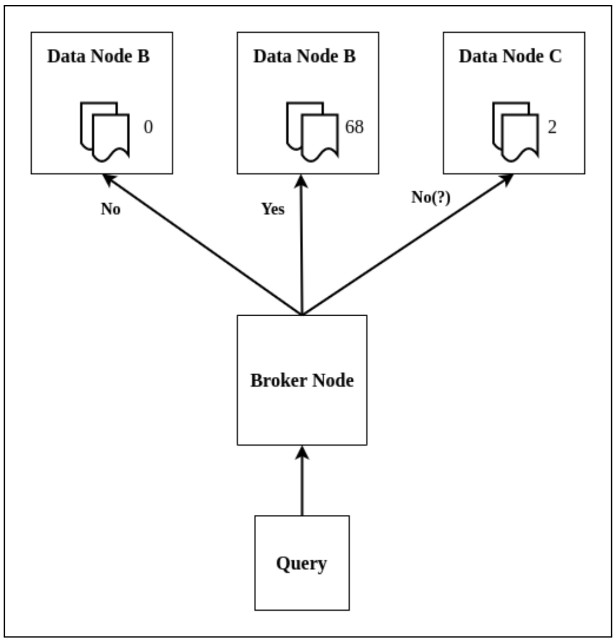

Figure 2: A schematic show of Shard Selection Problem

A proper shard selection will discard Node A and Node C, since shards in both of the nodes contain little or no relevant document and route the query only toward Node B.

When using the shard selection technique, the search relevance might be negatively impacted because some shard results might be lost due to the shard selection. Also, the shard selection process might be based on user requirements. For example, User A might need fast search results compromising some relevance in the data while User B might need high relevant data even though search time is slow. This research is an attempt to implement, analyze and optimize different Shard Selection along with the optimization of shard-related Elasticsearch parameters.

2. Literature Review

Scaling and Optimization of Distributed Systems and Big Data Technology [2, 3, 4] has been the major topic of interest since the early 90s. Among those, Shard related optimization is one of the major interests in the distributed system. Hence, many researches are being performed in a distributed search engine for the shard selection. For instance, the query-routing approach [5] is one of them. This approach explains how a query is routed to nodes for the relevant retrieval of the results.

There are many shard selection algorithms proposed till now. Collection Retrieval Inference Network (CORI) [6] was considered to be the first successful shard selection algorithm among them introduced in the mid-90s. It is based on Lexicon algorithms and based on a probabilistic model for information retrieval called a Bayesian network. Bayesian networks have been refined a lot since then.

HighSim [7] is considered as the best algorithm among a range of lexicon shard selection algorithms. It is an optimistic assumption-based algorithm. It assumes that a single document of the shard might contain all the terms from a query.

Other different types of algorithm than Lexicon Algorithms are the Surrogate algorithms [7, 8]. It was developed originally to work in distributed uncooperative search engines.

Best-N algorithm [8] is one of the Surrogate algorithms where the goodness for each term is calculated for each document.

Another surrogate algorithm developed for an uncooperative environment is ReDDE [9]. This algorithm is considered as benchmark for the shard selections. In this algorithm, a centralized index called CCI at the broker node is formed which contains sampled documents from all the shards in the data node. When a query from the user arrives at the broker node, estimation is done on the number of documents relevant to the query of each shard, and shards are ranked accordingly. The number of documents relevant is given in the equation 1.

$$Rel(S_{i,q}) = \sum_{d \in s_{i\_sampl}} P(Rel|d) \times \frac{|s_i|}{|s_{i\_sampl}|} \tag{1}$$

$Rel(S_{i,q})$ = Number of document relevant to query q in shards document-collection S_i

q = query performed
$P(Rel|d)$ = The estimated probability of relevance for document d to query q in CCI
S_{i_sampl} = set of sample document from shard S_i

The central rank of the document is approximated using the formula given in the equation 2.

$$rank\_central(d_i) = \sum_{rank\_samp(d_j)<rank\_samp(d_i)} \frac{|S_j|}{S_{j\_sampl}} \tag{2}$$

$P(Rel|d)$ is calculated as given in the equation 3.

$$P(Rel|d) = \begin{cases} \gamma, & \text{if } rank\_central(d) < \beta \times |S_i|. \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

And, the estimated relevance of given shard S_i on query q can be found with the goodness-score as given in the equation 4.

$$goodness(S_{i,q}) = \frac{Rel(S_i, q)}{\sum_j R(S_{j,q})} \tag{4}$$

Scoring Scaled Samples for Server Selection (Sushi) [10] is one of the latest surrogate algorithms. Sushi rank shards according to the centralized sample index based on two steps Rank adjustment and curve-fitting. The Rank Adjustment formula for Sushi is given in the equation 5.

$$rank\_adjusted(d) = (rank\_sample(d) + 0.5) \times \frac{|c|}{|S_c|} \tag{5}$$

Which shows the ratio of the size of the shard they belong to i.e. $|c|$ to the size of the sample from that shard $|S_c|$.

SHiRE [11] is another surrogate algorithm concept that utilizes a centralized sample index (CSI). In these algorithms, a bottom-up traversing hierarchy is followed for shard ranking. Voting to the shard is performed when a document is fetched from the shard. The formula for voting is given in the equation 6.

$$Vote(d) = S \times B^{-U} \tag{6}$$

S = score of the document given from the CSI ranking
B = exponential base
U = level at which the document was found in the hierarchy

In the Lexicon SHiRE (lex-s), the concept of lexical similarity between sample documents is used. It uses similarity to construct the hierarchy. On the other hand, in the Connected SHiRE (conn-s), shard-membership of the documents is used to construct the hierarchy.

Ranked SHiRE (rank-s) is considered the simplest hierarchy among the SHiRE family.

Some of the earlier research was done focusing on the performance and optimization of the above algorithms. In the paper [12], comparative performance analysis is done using Redde, Rank-s, and CORI shard selection algorithms. In the paper [13], a shard selection plugin called SAFE was developed that takes and analyzes ReDDe, HighSim, Sushi, and Rank-S to improve the performance. In the paper [14], resource selection, score normalization, and result merging techniques for small documents are analyzed. In the paper [15], a quantitative model of shard, placement strategy of the shard, and local balancing were considered for the optimization. Similarly, in the paper [16], a similar approach for Auto-Sharding is done on another NoSQL technology MongoDB.

3. Methodology

The research is an approach to analyze and optimize the existing shard selection algorithm along with the shard-related parameters. The work involves the implementation, analysis, and optimization of existing three popular shard selection algorithms namely called ReDDE, Sushi, and Rank-S along with shard-related parameters.

3.1 Data Set

Table 1 shows the datasets of Insider Threat Test Dataset(CERT V6.2) collected from Carnegie Mellon University are used for the experimentation purpose.

Table 1: Datasets for experimentation and parameter optimizations

File Name	Row Count	Data Size
email.csv	10994958	8.1 GB
http.csv	117025217	90.2 GB

Similarly, Shakespeare's play dataset used in the paper [12] is collected from the kaggle for validation purpose.

Table 2 shows the description of the dataset used for the validation.

Table 2: Datasets for validation

File Name	Row Count	Data Size
Shakespeare_data.csv	111397	10.2 MB

3.2 Parameters

Table 3 shows parameters that are used. They are initialized with the default values and updated during the process to get optimum parameters. The parameters directly affect either latency or document score or both of the Elasticsearch.

Table 3: Elasticsearch Parameters

Parameter	Description
index.number_of_shards	number of primary shards of an index
index.number_of_replicas	number of replicas of each primary shard
K	maximum number of shards allowed to select by a shard selection algorithm
indices.memory.index_buffer_size	allocation of heap memory (percentage or byte size)

3.2.1 Number of primary shards

It is the number of primary shards allocated for an Elasticsearch index. It has a direct impact on search latency.

3.2.2 Number of replicas

It indicates how many copies of the primary shard we allocate. For example, if we have 10 primary shards and 1 replica shard, we will have 20 shards in total. It has a direct impact on search latency.

3.2.3 Maximum Number of shards allowed to select

It is the maximum number of shards (among total primary and replica shard) which a shard selection algorithm can select. It has a direct impact on both search latency and document score.

3.2.4 Heap Memory Allocation

The heap memory is the amount of RAM allocated to the JVM of an Elasticsearch node. Heap memory allocation has a direct impact on search latency.

Impacts of above parameters are discussed in details on section 4.

3.3 Indexing and Searching

Data is indexed to Elasticsearch via logstash. The fields that need to be queried using the shard selection algorithm need to be specified in the configuration of the node. The algorithm only supports string fields. A precomputed CSI is processed using those fields during the sync operation between data nodes and broker nodes. Standard Analyzer is used for the experimentation process.

While searching using those specified fields, the CSI is again computed against the query to estimate and route to the relevant shards.

For searching, full-text searches are performed using Match Query on indices *http* and *email* to get the relevant documents. Match query is chosen because it is what mostly used in the full text searches. Several queries are executed for experimentation purposes. Following is an example of a query:

Query Type: Match Query
Query Index: http
Field Queried: content
Search String: “players coaches”

3.4 Architecture Design

An Elasticsearch cluster consisting of ten data nodes and two coordinating nodes(acting as broker nodes) is designed. For the experimentation purpose, as mentioned, the above two indices namely email and http are set such that each node contains a primary shard and a replica shard of both indices.

The architecture design is shown in the figure 3.

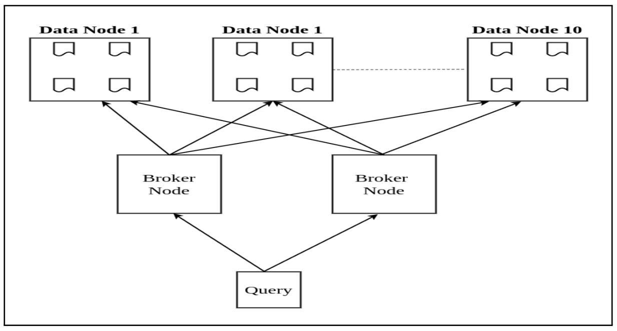

Figure 3: System Architecture Design for the implementation of Shard Selection Algorithms

3.5 Evaluation Metrics

Following are the evaluation metrics that are used to evaluate the shard selection algorithms.

1. Document Score
2. Query Latency
3. Precision at K (P@K)
4. Recall at K (P@K)

4. Results and Discussion

4.1 Experimental Setup

All the experiments were carried out in AWS Elasticsearch servers.

Table 4 shows the server description for the experimentation purpose.

Table 4: AWS Elasticsearch Server specification for the experimentation

Instance Type	Memory (GiB)	No. of Data Nodes	No. of Broker Nodes
r5.large.elasticsearch	16	10	2

Table 5 shows the server description for the validation purpose.

Table 5: AWS Elasticsearch Server specification for the validation

Instance Type	Memory (GiB)	No. of Data Nodes	No. of Broker Nodes
c6g.large.elasticsearch	4	5	1

4.2 Experimental Results and Discussion

The search query latency for different algorithms is shown in the figure 4.

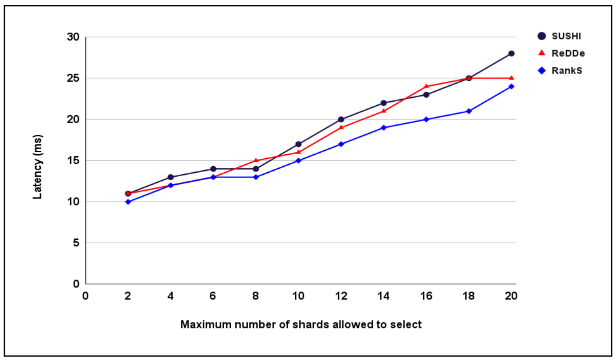

Figure 4: Latency (ms) against Maximum number of shards allowed to select for different shard selection algorithms

As the maximum number of shards the algorithm allowed to select increased, the latency is increasing. It is because the broker node has to reach the extra shard distributed over the data nodes. Rank-S is performing the best and SUSHI worst in term of latency

The average document score for different algorithms is shown in the figure 5.

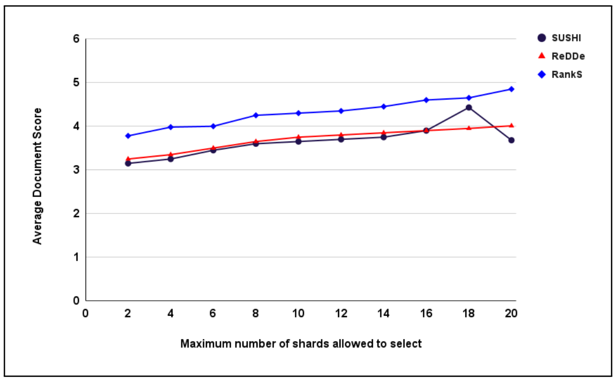

Figure 5: Average Document Score against Maximum number of shards allowed to select for different shard selection algorithms

As the maximum number of shards the algorithm allowed to select increased, the average Document score is increasing. It is because the broker node can select more relevant data from more shards. Rank-S is performing best and SUSHI worst in term of average document score.

Precision and Recall are calculated in reference to the relevant documents and retrieved documents in default search when no shard selection algorithm is applied. Precision@10, Precision@30, Recall@10, and Recall@30 for the query are shown in the Figure 6, Figure 7, Figure 8 and Figure 9 respectively.

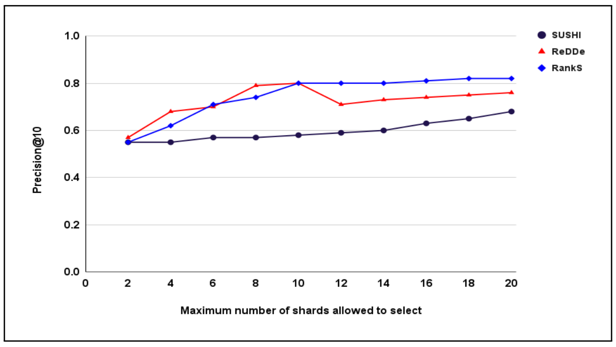

Figure 6: Precision@10 against Maximum number of shards allowed to select for different shard selection algorithms

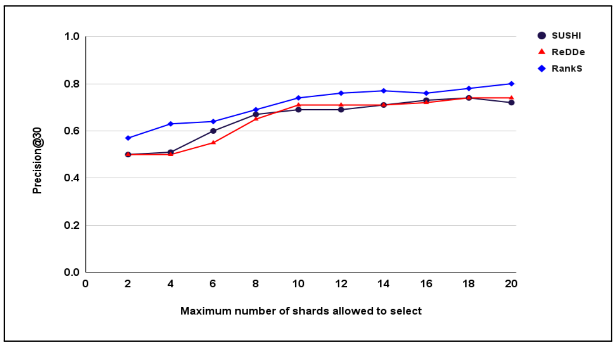

Figure 7: Precision@30 against Maximum number of shards allowed to select for different shard selection algorithms

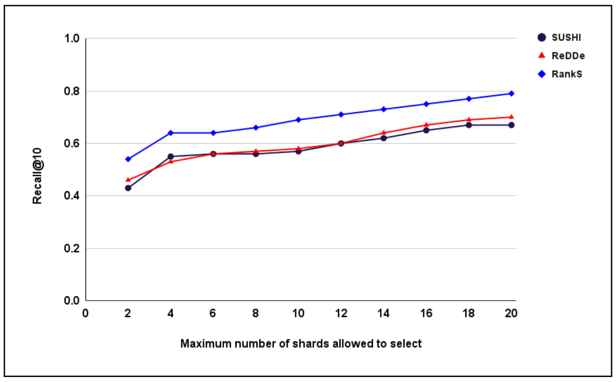

Figure 8: Recall@10 against Maximum number of shards allowed to select for different shard selection algorithms

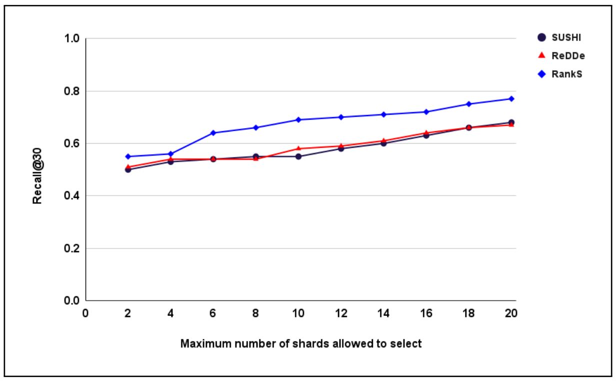

Figure 9: Recall@30 against Maximum number of shards allowed to select for different shard selection algorithms

For the Precision@10 and Recall@10, the top 10 documents are selected whereas for Precision@30 and Recall@30, the top 30 documents are considered. This works well because the users are mostly interested in top results from the query search. The recall and precision are improving as the maximum number of shards the algorithm allowed to select increased. It is because the broker node can get more relevant documents from more shards. Rank-S is performing best and SUSHI worst in terms of all the metrics.

The reason RankS performing better than all the algorithms is due to the extra step of voting by the document to the shard it was fetched from. The reason for SUSHI performing worst is still in the inspection and need to be analyzed using other data sets.

Along with the Shard Selection Algorithm, different shard-related parameters are analyzed and optimized. One of them is a maximum number of shards allowed to select by a shard selection algorithm which is shown in the given results.

Another parameter is the heap memory, the correlation for the Latency and Heap Memory for 16 GB memory and 122 GB memory nodes are shown in the figure 10 and figure 11 respectively.

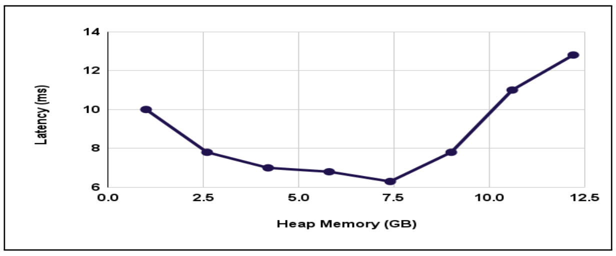

Figure 10: Latency (ms) against Heap Memory(GB) on the node having 16 GB memory

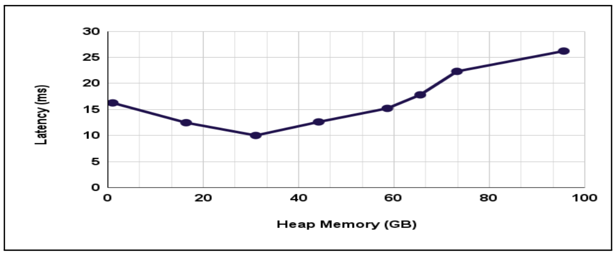

Figure 11: Latency (ms) against Heap Memory(GB) on the node having 122 GB memory

Heap memory seems to work fine if used less than half of the available memory or less than 31 GB, whichever is less. JVM Heap size above 31 GB seems to increase latency drastically. So, it's better to avoid it. Also, the heap memory allocation depends largely on required operations.

Another parameter to consider is the number of shards. Shard could be primary shard or replica shard. Here the number of replica used for each shard is 1. It means out of $2x$ shards, x shard is primary shard and x shard is replica shard. The relationship between search latency and the number of shards is given in the figure 12 and figure 13 for 25 GB Heap Memory and 15 GB heap memory respectively.

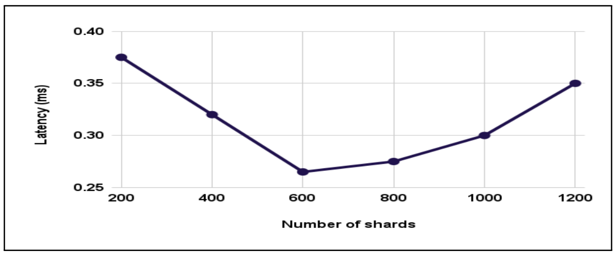

Figure 12: Latency (ms) against Number of shards on the node with 25GB Heap memory

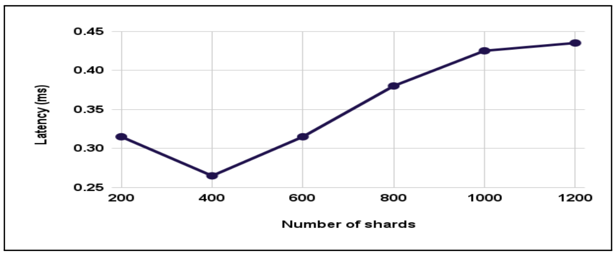

Figure 13: Latency (ms) against Number of shards on the node with 15GB Heap memory

The number of shards decreases the latency up to certain points, but after that, the number of shards becomes the overhead. Also, the Number of shards should be chosen according to the heap size of the Elasticsearch node. The higher the heap size, the higher could be the maximum number of shards chosen.

5. Validation Results and Discussion

Tables 6 and 7 show the average value of the validation results obtained performing the search operations. The validation is done in a different set of data, queries, and server configuration than that of the experimentation.

Table 6: Table showing average value of the validation results obtained from three queries

Methods	Search Latency (ms)	Average Document Score
RankS	23	4.32
ReDDe	25	3.45
SUSHI	26	3.65

As shown in table 6, Rank-S is performing best in terms of search latency and average document score.

Table 7: Table showing average value of the validation results obtained from three queries

Methods	P@10	P@30	R@10	R@30
RankS	0.74	0.72	0.57	0.67
ReDDe	0.59	0.62	0.59	0.57
SUSHI	0.61	0.65	0.62	0.59

As shown in table 7, Rank-S is also performing best in terms of Precision@10, Precision@30, Recall@10 and Recall@30.

6. Conclusion

Existing shard selection algorithms SUSHI, ReDDe, and Rank-S are implemented and analyzed in Elasticsearch along with the optimization of shard-related parameters.

As shown from the above results, more the number of shards allowed to select more is the Document Score, Precision, and Recall at the cost of the more Latency. This is because the broker node has to work on extra shard gaining extra information about a given query at the cost of more time and resources.

From the above experiments, in terms of average latency, considering overall queries and the maximum number of shards allowed to select, Rank-S is performing 14.92% and 9.83% better than SUSHI and ReDDe respectively.

In terms of Average Document Score, considering overall queries and the maximum number of shards allowed to select, Rank-S is performing 21.68% and 5.488% better than SUSHI and ReDDe respectively.

Also, a relationship between heap memory and latency is established. An increment of Heap memory decreases the latency up to a certain limit. Heap memory above certain increases latency drastically. Also, the heap memory allocation depends largely on required operations.

Also, a relationship between the number of shards and latency is established. The number of shards decreases the latency up to certain points, but after that, the number of shards becomes the overhead. Also, the Number of shards should be chosen according to the heap size of the Elasticsearch node. The higher the heap size, the higher could be the maximum number of shards chosen.

References

[1] Zhuyun Dai, Chenyan Xiong, and Jamie Callan. Query-biased partitioning for selective search. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 1119–1128, 2016.

[2] Hrishikesh Karambelkar. *Scaling Big Data with Hadoop and Solr*. Packt Publishing Ltd, 2013.

[3] Mohamad Sobhie. Tuning of elasticsearch configuration-parameter optimization through simultaneous perturbation stochastic approximation algorithm. Master's thesis, 2019.

[4] Quentin Coviaux. Optimization of the search engine elasticsearch. Master's thesis, Universitat Politècnica de Catalunya, 2019.

[5] Rafal Kuc and Marek Rogozinski. *Elasticsearch server*. Packt Publishing Ltd, 2013.

[6] James P Callan, Zhihong Lu, and W Bruce Croft. Searching distributed collections with inference networks. In *Proceedings of the 18th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 21–28, 1995.

[7] Daryl D'Souza, James A Thom, and Justin Zobel. Collection selection for managed distributed document databases. *Information processing & management*, 40(3):527–546, 2004.

[8] Milad Shokouhi and Luo Si. Federated search". foundations and trends in information retrieval (ftir). *Foundations and Trends in Information Retrieval*, 2011.

[9] Luo Si and Jamie Callan. Relevant document distribution estimation method for resource selection. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 298–305, 2003.

[10] Paul Thomas and Milad Shokouhi. Sushi: Scoring scaled samples for server selection. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 419–426, 2009.

[11] Anagha Kulkarni, Almer S Tigelaar, Djoerd Hiemstra, and Jamie Callan. Shard ranking and cutoff estimation for topically partitioned collections. In *Proceedings of the 21st ACM international conference on information and knowledge management*, pages 555–564, 2012.

[12] Praveen M Dhulavvagol, Vijayakumar H Bhajantri, and SG Totad. Performance analysis of distributed processing system using shard selection techniques on elasticsearch. *Procedia Computer Science*, 167:1626–1635, 2020.

[13] Per Berglund. Shard selection in distributed collaborative search engines a design, implementation and evaluation of shard selection in elasticsearch. 2014.

[14] Ilya Markov and Fabio Crestani. Theoretical, qualitative, and quantitative analyses of small-document approaches to resource selection. *ACM Transactions on Information Systems (TOIS)*, 32(2):1–37, 2014.

[15] Zhanglong Wang and Yang Pi. An optimization strategy of shard on elasticsearch.

[16] Yimeng Liu, Yizhi Wang, and Yi Jin. Research on the improvement of mongodb auto-sharding in cloud environment. In *2012 7th international conference on Computer science & education (ICCSE)*, pages 851–854. IEEE, 2012.